Steven M. Hoffberg

From:	Steven M. Hoffberg [steve@hoffberg.org]
Sent:	Thursday, November 18, 2004 12:15 PM
To:	'Nguyen, Nga'

Subject: 09/599,163 hash.c

```
/* -----------------------------------------------------------------------
 * -----------------------------------------------------------------------
 * hash.c - implements the HashTable Object
 *
 ***************************************************************************
 * original copyright and authorship:                           *
 * Copyright 1988, 1989, 1990, 1991 Massachusetts Institute of Technology   *
 *                                                    *
 * RMS 6/15/87                                                    *
 * hash.c : maintains a simple in-core hash table of name/value pairs     *
 ***************************************************************************
 * modified by: David M. Oliver
 *           Center for Geometry Analysis Numerics and Graphics
 *           University of Massachusetts Amherst
 *           oliver@gang.umass.edu
 * dmo 4.93 - extensive re-write to support void pointers, other generic ops
 * -----------------------------------------------------------------------
 * -----------------------------------------------------------------------
 */

#include <stdio.h>
#include <string.h>

#define HASH_IMPLEMENTATION

#ifndef MALLOC
#define MALLOC malloc
#endif  /* MALLOC */
#ifndef FREE
#define FREE free
#endif  /* FREE */

struct HASHLIST {           /* your basic hash table entry */
        char        *name;   /* the english name (manages own storage) */
        void        *value;   /* the entry (manages own storage) */
        struct HASHLIST *next;       /* the next node in the list */
};

typedef struct HASHLIST HashNode, *HashNodePtr;

struct HASHTABLESTRUCT {
        char        *name;     /* the name of this hash table */
```

```
    int         size;        /* the number of slots in the table */
    HashNodePtr  *table;        /* the actual table; array is table[size] */
    int     numentries; /* the total number of entries in table */
    int     (*hashf)(); /* hashing function */
};

typedef struct HASHTABLESTRUCT Table, *HashTable;

#include "hash.h"
#include "basics.h"
#include "aadefs.h"                    /* basic Object definitions */


/* ------------------------------------------------------------------------
 * HashErr - error printing
 * ------------------------------------------------------------------------
 */

#ifdef LOGGING
#define HashErr(message) LogMsg(DEFAULT_LOG_LEVEL, message);
#else
#define HashErr(message) fprintf(stderr, message);
#endif /* LOGGING */

/* ------------------------------------------------------------------------
 * hash - the internal (default) hashing function
 * ------------------------------------------------------------------------
 */

PRIVATE int
hash(s, size)
    char        *s;
    int   size;
{
  int     hashval;

  if ( size == 0 )
   return(0);                    /* avoid divide-by-zero */

  for ( hashval=0; *s; )
   hashval += *s++;
  return( hashval % size );
} /* end of hash */

/* ------------------------------------------------------------------------
 * strsave - create space, copy a string
 * ------------------------------------------------------------------------
 */

PRIVATE char *
strsave(s)
    char        *s;
```

```
{
  char  *p;
  int   len;

  if ( s == NULL )
   return (char *) NULL;

  len = strlen(s);
  if ( len > 0 ) {
    if( (p = (char *) MALLOC(len+1)) != NULL )
     strcpy(p,s);
     return (p);
  }
  else
   return (char *) NULL;
} /* end of strsave */

/* ------------------------------------------------------------------------
 * HashTableCreate - create a HashTable Object instance
 * ------------------------------------------------------------------------
 */

PUBLIC HashTable
HashTableCreate(name, size, hashfun)
    char        *name;              /* optional name of the hash table */
    int   size;               /* if <=0, makes default size */
    int (*hashfun)();             /* user defined hashing function */
{
  HashTable ht;

  if( (ht= (HashTable) MALLOC(sizeof(Table))) == NULL )
   return (HashTable) NULL;

  ht->name = strsave(name);                    /* NULL if no name given */
  ht->numentries = 0;
  if (hashfun == NULL)
   ht->hashf = hash;                  /* default hash function */
  else
   ht->hashf = hashfun;

  /*
   * allocate an array, HashNodePtr[size]
   */

  if (size > 0) {
    ht->table = (HashNodePtr *)MALLOC(size * sizeof(HashNodePtr));
    ht->size = size;
  }
  else {
    ht->table = (HashNodePtr *)MALLOC(DEFAULT_HASHSIZE * sizeof(HashNodePtr));
    ht->size = DEFAULT_HASHSIZE;
```

```
    }
    return ht;
} /* end of HashTableCreate */

/* ---------------------------------------------------------------------
 * HashTableDestroy - deallocate HashTable Object instance and ALL its data
 * ---------------------------------------------------------------------
 */

PUBLIC int
HashTableDestroy(ht, freefunc)
    HashTable         ht;
    int (*freefunc)();
{
    int              i;
    HashNodePtr      np, next;

    if( ht == NULL || ht->table == NULL )
      return CL_ERROR;

    for( i=0; i < ht->size; i++ )
     for( np = ht->table[i]; np != NULL; ) {
        next = np->next;
        if( np->name != NULL )
            FREE( (char *) np->name );
        if( np->value != NULL )
            (*freefunc)(np->value);        /* call user defined dealloc routine */

        FREE( (char *) np );
        np = next;
      }

    FREE( (char *) ht->table );
    if( ht->name != NULL )
      FREE( (char *) ht->name );
    FREE( (char *) ht );                   /* give it a valid ptr ... please */
    return CL_SUCCESS;
} /* end of HashTableDestroy */

/* ---------------------------------------------------------------------
 * HashTableLookup - find an element in the HashTable
 * ---------------------------------------------------------------------
 */

PUBLIC void *
HashTableLookup(ht, name)
    HashTable         ht;
    char   *name;                          /* name of the node to find */
{
    HashNodePtr       np;
```

```
  if( !ht || !name ) {
    HashErr("HashTableLookup: table pointer is NULL\n");
    return (void *) NULL;
  }

  for(np = ht->table[(*ht->hashf)(name, ht->size)]; np != NULL; np = np->next)
   if( !strcmp(name, np->name) )
    return (np->value);                          /* found it */

  return (void *) NULL;                          /* didnt */
} /* end of HashTableLookup */

/* -------------------------------------------------------------------------
 * HashTableInstall - add an element to the HashTable
 * NOTE: if (char *)name is already in the hash table, HashTableInstall()
 *       REPLACES the existing value with the new value passed as a parameter
 *       UNLESS the passed value is (void *)NULL.
 * -------------------------------------------------------------------------
 */

PUBLIC int
HashTableInstall(ht, name, value)
    HashTable       ht;              /* the hash table to use */
    char     *name;            /* an english name */
    void        *value;          /* the entry */
{
  HashNodePtr       np;
  int           hashval;

  if( ht == NULL ) {
    HashErr("HashTableInstall: table pointer is NULL\n");
    return CL_ERROR;
  }
  if( name == NULL ) {
    HashErr("HashTableInstall: name pointer is NULL\n");
    return CL_ERROR;
  }

  if( (np = HashTableLookup(ht, name)) == NULL ) {     /* not yet defined */
    np = (HashNodePtr) MALLOC(sizeof(HashNode));
    if( np == NULL )
     return CL_ERROR;
    np->value = NULL;
    if( (np->name = strsave(name)) == NULL )
     return CL_ERROR;
    hashval = (*ht->hashf)(np->name, ht->size);
    np->next = ht->table[hashval];
    ht->table[hashval] = np;
  }
  else {                              /* ----- already defined ----- */
    if(( np->value != NULL ) && (value != NULL) )
```

```
        FREE( (char *) np->value );
    }

    if (value != NULL) {
        np->value = value;
        return CL_SUCCESS;
    }

    return CL_SUCCESS;
} /* end of HashTableInstall */

/* ----------------------------------------------------------------------
 * HashTableRemove - remove an element from the HashTable
 * NOTE: node data is NOT removed (the function HashTableDestroy() DOES).
 * ----------------------------------------------------------------------
 */

PUBLIC int
HashTableRemove(ht, name)
    HashTable        ht;
    char    *name;
{
    HashNodePtr        np, temp, prev;
    int            hashval;

    if (!name || !ht) {
        HashErr("HashTableRemove: table pointer or name is NULL\n");
        return CL_ERROR;
    }

    if ((np = HashTableLookup(ht, name)) != NULL ) {        /* found it */
        hashval = (*ht->hashf)(name, ht->size);

        if (np == ht->table[hashval]) {            /* no previous entries */
            prev = NULL;
        }
        else {                            /* find the previous */
            for (temp = ht->table[hashval];
                temp!=NULL && temp->next!=np;
                temp = temp->next );
            prev = temp;
        }

        /*
         * if previous, set prev->next to point over np;
         * if no previous, reset root of list
         */

        if (prev != NULL)
         prev->next = np->next;
        else
```

```
        ht->table[hashval] = np->next;

      if (np->name != (char *) NULL)
       FREE ((char *) np->name );

      FREE ((char *) np);

      return CL_SUCCESS;
    }
    return CL_ERROR;              /* not in table, so can't remove it */
} /* end of HashTableRemove */


/* -------------------------------------------------------------------------
 * HashTableApply - apply a function to (the data of) each element of HashTable
 * -------------------------------------------------------------------------
 */

PUBLIC int
HashTableApply(ht, func)
    HashTable          ht;
    int   (*func)();
{
   int                i;
   HashNodePtr        np;

   if (ht==NULL) {
     HashErr("HashTableApply: table pointer is NULL\n");
     return CL_ERROR;
   }
   if (func==NULL) {
     HashErr("HashTableApply: function pointer is NULL\n");
     return CL_ERROR;
   }

   for ( i=0; i<ht->size; i++ )
    for ( np = ht->table[i]; np!=NULL; np = np->next )
     if ( (*func)( ht, np->value ) )
       return CL_ERROR;

   return CL_SUCCESS;
} /* end of HashTableApply */


/* -------------------------------------------------------------------------
 * HashTableGetName - get a HashTable's (optional) name
 * -------------------------------------------------------------------------
 */

PUBLIC char *
HashTableGetName(HashTable ht)
{
   if (!ht)
```

```
    return (char *) NULL;
    else
    return ht->name;
} /* end of HashTableGetName */


/* ----------------------------------------------------------------------
 * HashTableGetNumEntries - get a HashTable's number of of members
 * ----------------------------------------------------------------------
 */

PUBLIC int
HashTableGetNumEntries(HashTable ht)
{
   if (!ht)
    return CL_ERROR;
   else
    return ht->numentries;
} /* end of HashTableGetNumEntries */


/* ----------------------------------------------------------------------
 * HashTableSetFunction - set a user defined hashing function.
 * NOTE: function is in the form: int hashf(char *str, int s), where "s" is
 * the size of the hash table.
 * ----------------------------------------------------------------------
 */

PUBLIC void
HashTableSetFunction(HashTable ht, int (*hashf)())
{
   if (!ht)
     return;
   else
     ht->hashf = hashf;
} /* end of HashTableSetFunction */


/* ----------------------------------------------------------------------
 * HashTableDumpTable - dump HashTable Object to file
 * ----------------------------------------------------------------------
 */

PUBLIC int
HashTableDumpTable(ht, fp, prt)
    HashTable   ht;
    FILE        *fp;
    int   (*prt)();
{
   HashNodePtr        np;
   int               i;

   if (ht==NULL ) {
     HashErr("HashTableDumpTable: table pointer is NULL\n");
```

```
       return CL_ERROR;
    }
    if ( fp==NULL ) {
       HashErr("HashTableDumpTable: file pointer is NULL\n");
       return CL_ERROR;
    }

    fprintf(fp,"%s\n%d\n", ht->name, ht->size);  /* table's name, buffer size */

    /* dump table elements calling user-defined function to dump value */

    for( i=0; i < ht->size; i++ )
     for(np = ht->table[i]; np!=NULL; np = np->next) {
       fprintf(fp,"%s\n", np->name);
       if((*prt)(fp, np->value)) {
            HashErr("HashTableDumpTable: error dumping data\n");
            return CL_ERROR;
       }
     }

    return CL_SUCCESS;
} /* end of HashTableDumpTable */

/* -------------------------------------------------------------------------
 * HashTableLoadTable - Load a HashTable Object from a file
 * -------------------------------------------------------------------------
 */

PUBLIC HashTable
HashTableLoadTable(fp, scn, hashfun)
    FILE          *fp;
    void   *(*scn)();
    int (*hashfun)();
{
   int     size;
   char     buf[INTERNAL_STRING_LENGTH];
   HashTable ht;
   extern char msgString[];

   if (!fp || feof(fp)) {
      HashErr("HashTableLoadTable: file pointer is NULL or EOF\n");
      return (HashTable) NULL;
   }

   /* get table name and size */

   fscanf(fp, "%s\n%d\n", buf, &size);

   /* manufacture an instance  */

   ht = HashTableCreate(buf, size, NULL);
```

```
    if (!ht) {
        HashErr("HashTableLoadTable: cant create hash table\n");
        return (HashTable) NULL;
    }

    ht->numentries = 0;

    /* set the hashing function */

    if (hashfun == NULL)
     HashTableSetFunction(ht, hash);
    else
     HashTableSetFunction(ht, hashfun);

    /* read data into table */

    while(!feof(fp)) {
        int i = 2, err = 0;
        void *value;

        value = (void *)NULL;

        err = fscanf(fp, "%s\n", buf);        /* get entry's name, then ... */
        if (err < 0) {
            HashErr("HashTableLoadTable: load file empty\n");
            return (HashTable) NULL;
        }

        value = (*scn)(fp);    /* ... call user-defined function to get value */

        if ((! *buf) && !value) {
            sprintf(msgString, "HashTableLoadTable: error scanning line %d\n", i);
            HashErr(msgString);
        }

        if (!HashTableInstall(ht, buf, value)) {
            HashErr("HashTableLoadTable: HashTableInstall failed\n");
            return (HashTable) NULL;
        }
        i++;
    }

    return ht;
} /* end of HashTableLoadTable */
```

Very truly yours,

Steven M. Hoffberg
Milde & Hoffberg, LLP
Suite 460
10 Bank Street

White Plains, NY 10606
(914) 949-3100 tel.
(914) 949-3416 fax
steve@hoffberg.org
www.hoffberg.org

11/28/2006